UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Sistema de ayuda a la diagnosis y prognosis de diferentes enfermedades aplicando técnicas de Machine Learning

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Javier Costa Rosa

Tutor: Juan Carlos Pérez Cortés
François Signol

Curso 2017-2018

Resum

Aquest treball està centrat en el desenvolupament d'una ferramenta *web* d'ajuda a personal metge, proporcionant-les la possibilitat de realitzar distints tipus de prediccions sobre determinades malalties, incrementant la quantitat d'informació a disposició a l'hora de prendre una decisió sobre un tractament, un diagnòstic o un pronòstic. Per a ser capaç de realitzar aquestes prediccions, la ferramenta fa ús de tècniques de *Machine Learning*, una branca d'estudi de la Intel·ligència Artificial. Aquesta ferramenta *web* es planteja des de zero: s'han de desenvolupar tots els components que la formen, tant la part del client (*frontend*) com la part del servidor (*backend*). Per a això, es fa ús del model *MEAN Stack*, un conjunt de tecnologies que tenen un llenguatge de programació en comú: JavaScript.

El present document mostra el desenrotllament que s'ha dut a terme per a construir la ferramenta *web*, passant per l'ús bàsic que s'ha realitzat d'algunes tècniques de *Machine Learning*, així com la seua implantació i la seua posada en marxa.

Paraules clau: diagnosis, prognosis, malalties, Machine Learning, ferramenta web

Resumen

Este trabajo está centrado en el desarrollo de una herramienta *web* de ayuda a personal médico, proporcionándoles la posibilidad de realizar distintos tipos de predicciones sobre determinadas enfermedades, incrementando la cantidad de información a disposición a la hora de tomar una decisión sobre un tratamiento, un diagnóstico o un pronóstico. Para ser capaz de realizar estas predicciones, la herramienta hace uso de técnicas de *Machine Learning*, una rama de estudio de la Inteligencia Artificial. Esta herramienta *web* se plantea desde cero: se deben desarrollar todos los componentes que la forman, tanto la parte del cliente (*frontend*) como la parte del servidor (*backend*). Para ello, se hace uso del modelo *MEAN Stack*, un conjunto de tecnologías que tienen un lenguaje de programación en común: JavaScript.

El presente documento muestra el desarrollo que se ha llevado a cabo para construir la herramienta *web*, pasando por el uso básico que se ha realizado de algunas técnicas de *Machine Learning*, así como su implantación y su puesta en marcha.

Palabras clave: diagnosis, prognosis, enfermedades, Machine Learning, herramienta web

Abstract

This work is focused on the development of a web tool to help medical staff, allowing them to make different types of predictions about some certain diseases, increasing the information when it comes to making a decision about a treatment, a diagnosis or a prognosis. In order to do that, the tool makes use of Machine Learning techniques, one of the fields of study of Artificial Intelligence. This web tool is arisen from scratch, which means that all of its components must be developed, both the client side (frontend) and the server side (backend). To make this easier, it is based in the MEAN Stack model, a set of technologies that are developed using the same programming language: JavaScript.

This document shows the development process followed in order to build the web tool, through the basic use of Machine Learning technologies and finally its setup and its startup.

Key words: diagnosis, prognosis, diseases, Machine Learning, web tool

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

Los avances en el campo de la Inteligencia Artificial (en adelante IA) y, más en concreto, en la rama de *Machine Learning* (en adelante ML) durante los últimos años han sido muy relevantes en casi cualquier ámbito y han tenido un gran impacto en nuestras vidas. Por ejemplo, sin estas técnicas no podríamos hacer búsquedas tan eficaces en Google, no tendríamos tan buenas recomendaciones en nuestra cuenta de Facebook o Spotify, y por supuesto no tendríamos disponible (o no de forma tan eficaz) los cada vez más famosos asistentes personales, como el Asistente de Google para Android, Siri para IPhone o Cortana para Windows.

Estas técnicas han revolucionado la forma de plantear los problemas, como decíamos, en casi cualquier ámbito. En concreto, en el campo de la medicina, es posible aplicar el uso de las técnicas y herramientas de ML para, en teoría, poder predecir en cierta medida la aparición de ciertas enfermedades y otros factores relacionados con las mismas como, por ejemplo, la respuesta del paciente a un tratamiento asignado.

Por otro lado, gracias a las tecnologías web, hoy en día cualquier tipo de aplicación se puede tener al alcance de la mano en cualquier momento y situación, pues la gran flexibilidad de las mismas hace que se pueda hacer uso de ellas indistintamente desde un ordenador, un teléfono móvil o, en general, desde cualquier dispositivo que sea capaz de conectarse a Internet.

En este proyecto se combinan ambos elementos. Por un lado, se dispone un servicio que realiza predicciones mediante técnicas de ML y, por otro, se tiene una aplicación web que hace uso de este servicio y permite al usuario realizar diferentes consultas mostrando como resultado las predicciones generadas por el mismo.

1.1 Motivación

El Instituto Tecnológico de Informática (ITI), dentro de su línea estratégica de salud y en colaboración con el hospital La Fe de Valencia y otros hospitales y centros de investigación sanitaria, realiza experimentos mediante técnicas de ML sobre distintas enfermedades en busca de características relevantes que puedan servir para, por un lado, confirmar las características de las muestras (pacientes) que se sabe que tienen relevancia en una enfermedad y encontrar nuevas de estas características (proceso denominado como selección de características) y, por otro, poder llegar a realizar predicciones sobre distintos factores relacionados con enfermedades haciendo uso de esta previa selección de características.

Este proyecto empieza a plantearse cuando el personal de La Fe comunicó a los investigadores del ITI que sería útil para ellos disponer de una herramienta con la cual

pudieran obtener distintas predicciones sobre distintos factores de una determinada enfermedad en tiempo real, y desde el mismo laboratorio o consulta donde analizan y realizan el seguimiento de los resultados correspondientes a los análisis clínicos y genéticos de sus pacientes.

1.2 Objetivos

El objetivo principal de este proyecto es proporcionar una herramienta a los médicos de los hospitales mediante la cual, introduciendo datos sobre los pacientes, puedan obtener un conjunto de estimaciones de predicción o diagnóstico para cada uno de los factores clave relacionados con una determinada enfermedad y junto con sus probabilidades asociadas, que sirva como ayuda a las difíciles decisiones que tienen que realizar a la hora de evaluar a los pacientes que padecen ciertas enfermedades y a todos los factores que afectan a los mismos en cuanto a la enfermedad en sí.

Por lo tanto, los objetivos de este proyecto son:

- Desarrollar una aplicación web accesible desde cualquier parte que usará el médico para realizar las predicciones y que sea capaz de hacer uso del servicio de predicciones, con todo lo que ello conlleva (la parte *frontend*, la parte *backend*, comunicación entre el *frontend* y el *backend*, etc.)
- Desarrollar un servicio basado en técnicas de ML que calcule las predicciones correspondientes a un conjunto definido de tareas, basándose en modelos preentrenados fruto de la investigación ya realizada sobre diversas enfermedades que ha permitido encontrar las características más relevantes dentro de la información proporcionada por los análisis genéticos, así como los modelos matemáticos que permiten realizar los cálculos necesarios para estimaciones en relación con cualquier tarea de interés definido para cada enfermedad.

Durante el desarrollo del proyecto se escoge como caso de uso un tipo de cáncer de sangre denominado Leucemia Mieloide Aguda. Como principal objetivo se estudia una tarea concreta consistente en predecir la respuesta a un tratamiento de inducción con dos posibles resultados: remisión o no remisión de la enfermedad.

1.3 Estructura de la memoria

Lo primero que se realizará es un estudio del estado del arte en relación con este tipo de herramienta, así como con el avance actual en la aplicación de técnicas de ML aplicadas al ámbito del diagnóstico y el pronóstico de enfermedades. Se realizará un análisis previo de lo que se quiere obtener, justificando cada elección. Después se incluirá un apartado detallando la arquitectura y la estructura del *software* a desarrollar, tanto de la herramienta en sí como del servicio de predicción. Posteriormente se explicará en detalle cómo se ha desarrollado el sistema y su puesta en marcha y, por último, se describirán las conclusiones finales.

CAPÍTULO 2
Estado del arte

Hoy en día, la IA está en todas partes y su aplicación en muchos ámbitos de nuestra vida está en plena expansión. Cuando se realiza una búsqueda por Internet se está acudiendo a ella, también se encuentra en los asistentes de nuestros dispositivos móviles, en las recomendaciones de amigos de Facebook, etc. Cada día, dado que cualquier dispositivo con una mínima capacidad computacional puede integrar IA, más aplicaciones con esta tecnología llegan a los usuarios y se avanza en el desarrollo de herramientas cada vez más sofisticadas y con un potencial cada vez más amplio.

La disciplina que más resuena dentro del campo de la IA es, sin duda, la de ML, la cual se dedica a desarrollar técnicas que permiten a un programa aprender en base a la experiencia de haber sido expuesto a una determinada información, de forma que se crean lo que se conoce como modelos mediante los cuales, posteriormente, se pueden generar predicciones sobre una nueva información entrante.

En concreto, una de las aplicaciones de esta disciplina es el ámbito de la medicina, en el cual es posible diagnosticar de forma precoz y predecir en cierta manera la aparición de enfermedades, poder predecir la evolución de un paciente a una enfermedad concreta, predecir la respuesta del paciente a un tratamiento asignado para tratar una enfermedad, etc.

Por ejemplo, la empresa *Genomic Prediction*[1], dispone de una forma de predecir, en el contexto de la realización de una fecundación in vitro, qué futuros embriones de los que se dispone serán más probables de desarrollar diabetes tipo 1 entre otras enfermedades, en base a un análisis genético de los mismos mediante dichas técnicas de ML[1]. Así, tanto los padres del futuro bebé como los médicos pueden descartar los que son más propensos a padecer estas enfermedades y escoger los que menos lo son.

Otro ejemplo lo encontramos en la empresa *Myriad*[2], la cual dispone de muchos productos relacionados con la predicción de riesgos en función de la información genética para 7 tipos de enfermedad distintos. Por ejemplo, dispone de una aplicación mediante la cual se estudia el riesgo de 8 tipos de cáncer hereditarios analizando, también, los datos genéticos de los pacientes.

En el contexto del ITI, también se estudian un conjunto de enfermedades y la forma de diagnosticar y pronosticarlas de la mejor forma posible, identificando las variables clínicas o genéticas que influyen en dichas enfermedades. Entre ellas, se encuentra el cáncer de mama, sobre la cual, fruto de las investigaciones descritas, se ha desarrollado una aplicación llamada DMScan[3], que facilita la segmentación del tejido denso de la mama a los

[1] https://genomicprediction.com/
[2] https://myriad.com/
[3] http://dmscan.iti.upv.es/

radiólogos, dado que es un biomarcador influyente en la aparición de esta enfermedad, y que ofrece un pronóstico de la aparición de este tipo de cáncer a los 2 años usando la información de la mamografía. Además del cáncer de mama, en el ITI también se estudia la enfermedad de la Leucemia Mieloide Aguda. En concreto, se buscan las variables influyentes para dos tipos de predicciones: la respuesta al tratamiento de inducción, y la supervivencia libre de enfermedad una vez acabado este tratamiento.

Lo que se propone dentro del ITI y como TFG es crear una herramienta genérica, es decir, no centrada en una determinada enfermedad concreta, que permita realizar determinadas predicciones al personal médico sobre distintas enfermedades. Esta herramienta, como novedad respecto a las ya vistas, estaría disponible desde cualquier dispositivo y en cualquier parte. Al proponerse como aplicación *web*, el personal médico podría acceder desde el ordenador de su consulta, desde su teléfono móvil o desde cualquier otro dispositivo compatible, mediante usuario y contraseña, a la herramienta en forma de página *web*. Así, en la misma consulta con el paciente o en cualquier momento que considere, puede acceder a dicha herramienta y realizar las predicciones que considere oportunas, recibiendo la respuesta en tiempo real.

CAPÍTULO 3
Análisis del problema

En esta sección se realizará un minucioso estudio del problema: qué es lo que necesita para llevar a cabo sus funciones, cuáles son los requisitos del sistema en base a los necesidades del usuario y que estrategias se llevan a cabo en cuanto a la seguridad del mismo y en cuanto al cumplimiento de las normativas vigentes.

3.1 Sistema

En base a los objetivos definidos y a la motivación del proyecto descritos en el capítulo 1, y como ya se ha comentado en dicho capítulo, el sistema está dirigido a un usuario específico. En concreto, el usuario del sistema será el médico que realizará la predicción desde su consulta o desde donde lo necesite. Para realizar esta operación deberá proporcionar al sistema datos tanto clínicos como géneticos de un paciente, en función del tipo de predicción que realice. Por tanto, el usuario necesita que se le proporcione un formulario mediante el que pueda introducir estos datos.

Como se busca que el sistema sea accesible desde cualquier lugar, puesto que se supone que la figura del médico investigador puede querer o necesitar poder realizar una determinada predicción en cualquier momento de su trabajo, se ha decidido desarrollar el mismo como una aplicación *web*. Así, el usuario puede acceder al sistema desde cualquier dispositivo y realizar la predicción que busca, puesto que se pretende que la página *web* también sea adaptativa. Esto es, que se adapte a cada dispositivo en cuanto al apartado visual como, por ejemplo, que los elementos de la misma se reubiquen si se visita desde un dispositivo con una pantalla más pequeña, como puede ser un teléfono móvil.

Como la aplicación *web* podrá ser accesible desde cualquier parte, se debe evitar que cualquier persona pueda acceder al sistema. Por tanto, se precisa de un sistema de autenticación mediante el cual los usuarios puedan iniciar sesión para acceder al sistema y disponer de una configuración personal como, por ejemplo, sus pacientes asignados. Esto conlleva disponer de una base de datos en la que guardar los usuarios, su configuración y los pacientes.

Es interesante también disponer de varios tipos de usuarios. Uno de ellos sería el médico y otro podría ser un administrador que asigne los pacientes a los médicos, o cualquier otro tipo de tarea de gestión. En el ámbito en el que nos encontramos, se descarta un proceso de registro en la aplicación *web* al ser de uso privativo, siendo usuarios de la misma únicamente el o los hospitales que adopten el sistema. Por tanto, el registro de usuarios del sistema se delega en usuarios de tipo administrador.

También se necesita una forma de realizar las predicciones. Para minimizar el acoplamiento del sistema, lo más indicado es implantar este sistema de predicción en un

servidor aparte, el cual formaría parte del conjunto de componentes del sistema, pero sería un módulo independiente del resto de dichos componentes. Esto implica la creación de un servidor en el que únicamente se realicen esas predicciones, y al que se le habrá de realizar una petición comunicándose de una forma específica con él para obtener una predicción. Por otro lado, para garantizar que se pueden realizar predicciones fiables es necesario construir un sistema predictor adecuado. Esto implica, principalmente, disponer de modelos preentrenados, lo cual implica, a su vez, que dichos modelos deberán ser guardados y estar disponibles en todo momento de alguna forma para que el sistema de predicción pueda acceder a ellos cuando lo necesite. Esta forma de guardarlos puede ser en la misma base de datos en la que se va a guardar la información del sistema.

Resulta interesante también el estudio de la posibilidad de almacenar las predicciones realizadas, con el fin de un análisis posterior en forma, por ejemplo, de estudio de regresión y evolución de un determinado paciente para, además de ayudar al médico en su labor revisando el progreso del paciente en comparación con lo que las predicciones dictan, tener alguna forma de retroalimentación para poder conocer el nivel de certeza el cual finalmente pueden llegar a alcanzar las predicciones. Estas predicciones realizadas serían guardadas también en la base de datos del sistema.

Dado que la base de datos y el servicio de predicción no deben ser accesibles por cualquier persona, puesto que es para uso específico del sistema, se precisa de un componente intermediario que realice la conexión con estos componentes. Este intermediario (al cual en adelante se le identificará por el término en inglés *middleware*) ocultará la existencia de dichos componentes al usuario, de forma que aumente sustancialmente la seguridad de ambos componentes, al ofrecer así la posibilidad de que estos componentes estén en una red interna que haga imposible acceder a ellos desde el exterior sin pasar por dicho intermediario.

Teniendo en cuenta todo lo expuesto, se puede concluir que, para llevar a cabo el desarrollo de la herramienta, se han de implementar un conjunto de componentes de manera que formen un sistema *web*, siendo uno de ellos el que se encargará de realizar las predicciones. En concreto, y a modo de resumen:

- Un *frontend*, el cual se encargará de los mecanismos para representar el sistema de forma visual, de forma que el usuario pueda interactuar con el mismo
- Un *middleware* que interconecte el *frontend* con el resto de componentes
- Un servidor cuyo cometido sea realizar una predicción en base a una petición entrante
- Una base de datos donde guardar la información del sistema y los modelos para la predicción

3.2 *Machine Learning*

Se conoce como *Machine Learning* la rama de estudio de la IA dentro del campo de las ciencias de la computación cuyo objetivo es dar a los computadores la habilidad de aprender, esto es, de generalizar comportamientos a partir de una información determinada que se les enseña previamente, creando modelos de esta información mediante los cuales se puede, posteriormente, realizar una predicción de una nueva información entrante.

Es importante aclarar que en este proyecto se van a usar técnicas de ML para realizar las predicciones, pero no se va a profundizar en exceso en cuanto a cómo funcionan estas técnicas.

En este proyecto nos encontramos con un problema típico de ML supervisado: disponemos de un conjunto de muestras, un conjunto de variables y un conjunto de etiquetas de clase. En el ámbito del proyecto, estos se corresponden con el conjunto de pacientes, el conjunto de datos de los mismos y las posibles predicciones, respectivamente. Todos estos datos son proporcionados por los médicos quienes disponen de esta información en base a la investigación y al seguimiento que realizan de diversos pacientes.

Como se puede observar gráficamente en la figura 3.1, durante un proceso básico de predicción mediante este tipo de técnicas se siguen los pasos descritos a continuación:

1. Llegan nuevos datos al sistema los cuales contienen variables del problema.
2. Se realiza una transformación de estos variables, convirtiéndolas en características
3. Se realiza la imputación de datos faltantes a partir de los datos de entrenamiento
4. Se consulta el modelo predictivo con estos datos, el cual emite la predicción
5. Se obtiene la predicción

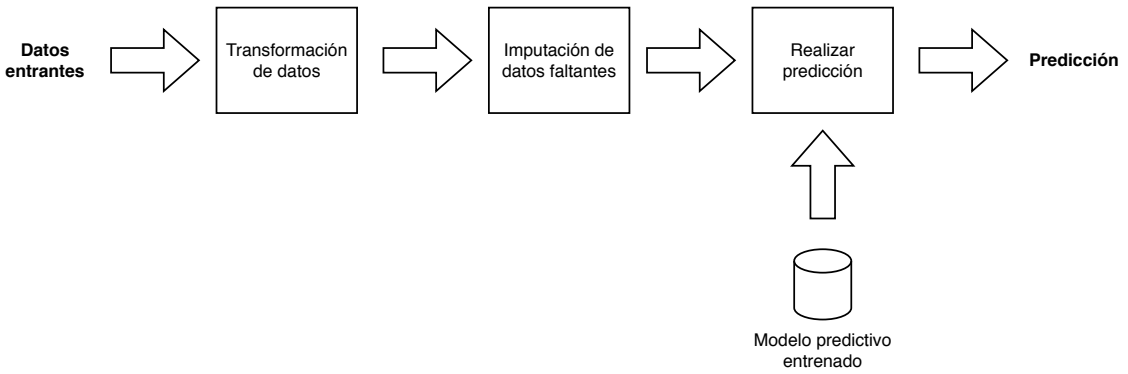

Figura 3.1: Proceso de predicción en un sistema de ML

A continuación se explica y detalla cada paso de los que se ha hablado:

3.2.1. Variable y característica. Transformación

La transformación es el proceso de aplicar una determinada operación a una variable con el objetivo de modificar su valor. Como resultado, obtenemos una característica. La transformación se realiza con todas las variables de todas las muestras, de forma que al final se obtiene un conjunto de datos normalizado y preparado para entrenar un modelo o realizar una predicción.

La transformación de variable a característica es, en este caso, un paso obligatorio puesto que cada variable puede tener un rango de valores. Por ejemplo, una variable continua como un porcentaje tendrá un rango de valores desde el 0 hasta el 100, mientras que si la variable es categórica tendrá un rango de valores discretos y limitados. Por tanto, una normalización de estos datos es necesaria para poder trabajar con ellos.

3.2.2. Imputación de datos faltantes

La imputación de datos faltantes es el proceso de obtener, a partir de una variable de la cual no conocemos su valor, una característica. Existen diversos métodos para esto, siendo uno de los más comunes el de obtener la mediana o la moda de los distintos valores de la misma característica entre todas las muestras. De esta forma, se obtienen estimaciones verosímiles del posible valor aun sin conocerlo.

Este paso es obligatorio puesto que una vez el médico se encuentre en la consulta con el paciente, puede que no tenga todos los datos del mismo, sino una parte de ellos. Por tanto, es también interesante poder ofrecerle la posibilidad de realizar una predicción en base a los datos de los que disponga en un determinado momento.

3.2.3. Modelo predictivo

Un modelo predictivo debe ser entrenado previamente mediante un conjunto de características de las muestras junto con sus respectivas etiquetas de clase y, sobre la cual, una vez entrenada, se pueden realizar consultas con nuevos valores de esas características para obtener una predicción.

En este proyecto, los modelos predictivos preentrenados vienen proporcionados por el ITI en base a sus investigaciones previas. Por tanto, solo se realizarán las consultas a dichos modelos con el fin de obtener las predicciones, sin especificar el entrenamiento de los mismos.

3.2.4. Muestras

Es el conjunto de variables con los que se entrena un modelo. En este proyecto, cada paciente se corresponde con una muestra.

3.2.5. Etiquetas de clase

Es el conjunto de datos que contiene las posibles predicciones que se pueden dar. En este proyecto, a cada paciente le corresponde una etiqueta.

3.3 Seguridad

Dado que la página *web* será accesible desde cualquier parte y por cualquier persona, el primer paso para garantizar una mínima seguridad es disponer de un sistema de autenticación como ya se ha comentado. Por otra parte, el sistema deberá admitir comunicaciones por un canal seguro, esto es, mediante un certificado que garantice una conexión segura, ya que se transmitirá muy probablemente información sensible como las contraseñas de los usuarios cuando inician sesión.

Otro de los pasos para aumentar la seguridad es la idea del *middleware* comentado anteriormente, pues este oculta la existencia de la base de datos y del servidor de predicción, permitiendo que estos estén ubicados en una red interna a la que únicamente dicho *middleware* pueda tener acceso, de manera que sea imposible acceder a estos componentes desde el exterior. Así, dispondremos de una pasarela la cual puede ofrecer únicamente los datos que realmente deban ser públicos o que interesen al usuario.

3.4 Protección de datos

Los datos de los pacientes que se manejan en este problema son una información extremadamente sensible que está protegida por la legislación vigente. Para evitar este tipo de problemas, se asume que el hospital dispone de una base de datos de pacientes, de cada cual existe un código identificador único con el que se trabajará. El médico podrá saber de qué paciente se trata mediante este código identificador, darlo de alta en el sistema y trabajar con él sin que se guarde información directa del mismo.

CAPÍTULO 4
Diseño del sistema

En esta sección se detalla el diseño de la estructura y arquitectura del sistema *web*, así como las tecnologías usadas para cada uno de sus componentes, cómo se comunican entre ellos y cuál es su principal papel dentro del sistema. Los detalles técnicos en cuanto a la implementación del diseño descrito en este capítulo se concretarán en el siguiente capítulo (5).

4.1 Diseño

El funcionamiento básico del sistema es el siguiente:

1. El *frontend*, para satisfacer la petición del usuario, solicita datos al *middleware*
2. El *middleware* recibe una petición de un cliente y se comunica con el servidor de predicción o con la base de datos, dependiendo del tipo de la misma
3. En caso de que la petición sea de predicción, el servidor de predicción se comunica con la base de datos para obtener los datos necesarios para realizar la misma (El funcionamiento del servidor de predicción se especifica en la sección 4.4.3)
4. El *middleware* obtiene la respuesta del servidor de predicción o de la base de datos y responde con los datos obtenidos al cliente que realizó la petición
5. El *frontend* procesa los datos y los muestra de forma gráfica al usuario

El diagrama de clases representando las entidades existentes en el sistema y las relaciones entre ellas se representa en la figura 4.1. A continuación se detalla cada entidad y sus atributos:

- *User*: Representa al usuario. En este caso, solo tiene un nombre de usuario y una contraseña, además de un campo que indica si es administrador o no
- *Patient*: Representa un paciente. Solo tiene un campo, *code*, un identificador mediante el cual el médico puede saber de qué paciente se trata, pero que anonimiza al paciente de cara al sistema ya que no se mantiene ningún dato personal
- *Prediction*: Representa una predicción que se ha llevado a cabo. Tiene un paciente y una tarea asociada, la fecha en la que se obtuvo y el resultado de la predicción

- *Task*: Representa una tarea de predicción. Tiene un título (qué es lo que hace la tarea), un nombre (un identificador parecido a "T1R1b2", usado por el ITI para identificar las tareas), una descripción (explica con más detalle qué realiza esa tarea), una etiqueta (suele ser la enfermedad con la que está relacionada), un diccionario que contiene como claves las posibles predicciones de forma identificativa y como valores las mismas predicciones de forma entendibles por un ser humano, unas variables asociadas y un modelo asociado
- *Variable*: Representa una variable del paciente. Tiene un nombre (el nombre de la variable normalizado para que sea único), una descripción, una designación (el nombre original de la variable), la unidad en que se mide, el tipo de variable que es y las categorías posibles que puede tener
- *Model*: Representa un modelo. Tiene un clasificador (el que realiza la predicción), un X (las muestras con las que se ha entrenado el modelo) y un y (el conjunto de etiquetas). Cabe destacar que tanto el clasificador como las muestras y el conjunto de etiquetas están serializados para poder almacenarlos por lo que, en el momento en que se requiera utilizarlos, se habrán de procesar previamente realizando el proceso inverso

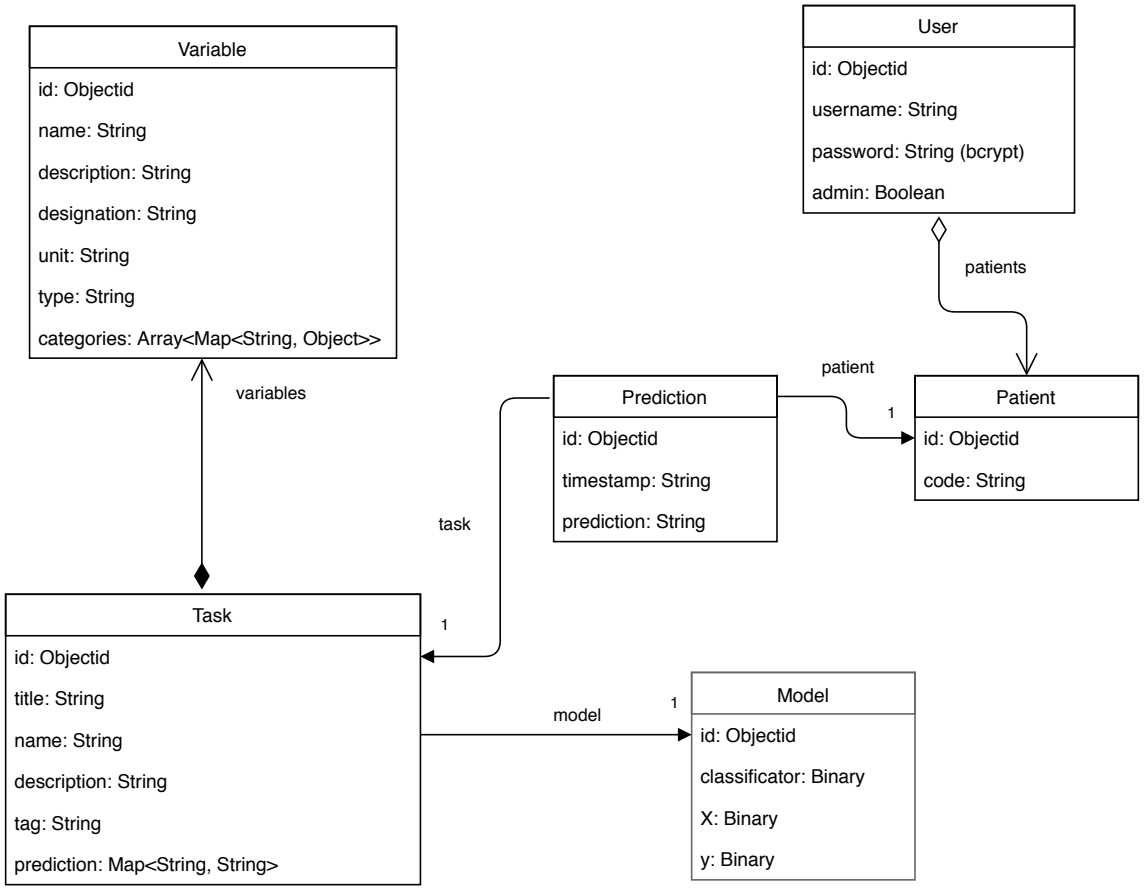

Figura 4.1: Diagrama de clases del sistema

4.2 Arquitectura

La arquitectura del sistema está basada en un sistema REST (estilo arquitectónico basado en el protocolo cliente-servidor donde cada petición realizada por el cliente es independiente de cualquier otra y donde el servidor establece una API de comunicación[2][3]) de 3 capas (capa de presentación, capa de lógica y capa de persistencia) con un *middleware* como servidor *web*. La capa de presentación se corresponde con el *frontend* (cliente), la capa de lógica se corresponde con el servidor de predicción y, por último, la capa de persistencia se corresponde con una base de datos. El *middleware* hará la función de servidor *web* que establece una API para poder ser usado, el cual se conectará al servidor de predicción o a la base de datos según se requiera, haciendo estos componentes invisibles para el cliente.

Dado que se trata de un sistema REST, tanto el cliente como el servidor son módulos independientes. Gracias a esto, el cliente o el servidor se puede cambiar por otro componente que se comporte de la misma forma y el sistema seguirá funcionando. Por tanto, no dependen el uno del otro para poder funcionar correctamente de manera independiente. Así, el *frontend* puede desarrollarse en la tecnología que se quiera o podrán hacerse cambios en el mismo pero podrá seguir comunicándose con el *middleware* mientras sepa cómo usar su API y viceversa, sin afectar a otros componentes. El *middleware* también podrá comunicarse con un servidor de predicción y con una base de datos cualesquiera de la misma forma.

En la figura 4.2 se representa de forma gráfica la arquitectura del sistema, así como las comunicaciones entre los distintos componentes que lo conforman.

4.3 Tecnologías usadas

El sistema se basa en el modelo que se conoce como ***MEAN** Stack*[1], el cual usa las siguientes tecnologías:

- **M**ongoDB[2], un sistema de bases de datos NoSQL
- **E**xpress[3], un *framework* para NodeJS que proporciona características muy robustas para crear aplicaciones web fiables de forma rápida y sencilla
- **A**ngular[4], un *framework* para la programación de *frontends*. En este caso, se usará la versión 4.
- **N**odeJS[5], un entorno multiplataforma que ejecuta JavaScript en el lado del servidor, para la programación del *middleware* junto con Express

Sumado a esto, se dispone de un servidor de predicción desarrollado en Python.

En la sección 4.4 se detallan las tecnologías usadas para cada componente de manera individual.

En cuanto a las comunicaciones entre los componentes, se adopta como estándar el uso de JSON para transmitir la información en cualquier tipo de comunicación:

[1] http://www.mean.io/
[2] https://www.mongodb.com
[3] https://expressjs.com/
[4] https://angular.io/
[5] https://nodejs.org/en/

- La comunicación entre el *frontend* y el *middleware* se realiza mediante peticiones HTTP, al ser un servicio REST. El cuerpo de cada respuesta consiste en un JSON que contiene los datos pertinentes
- La comunicación entre el *middleware* y el servidor de predicción se realiza mediante un socket ZMQ, y se transportan los datos también mediante un JSON
- La comunicación entre el *middleware* o el servidor de predicción y la base de datos se realiza por medio de las librerías usadas en cada uno de ellos (En la sección 4.4 se detallan las tecnologías usadas para cada componente, así como las librerías de las que se habla). A nivel interno, estas librerías gestionan la comunicación con el servicio de la base de datos

Para más detalles en cuanto a los datos transportados en las comunicaciones entre los componentes, véase el anexo A.

En la figura 4.2, además de la arquitectura del sistema, se representan las tecnologías usadas para cada componente.

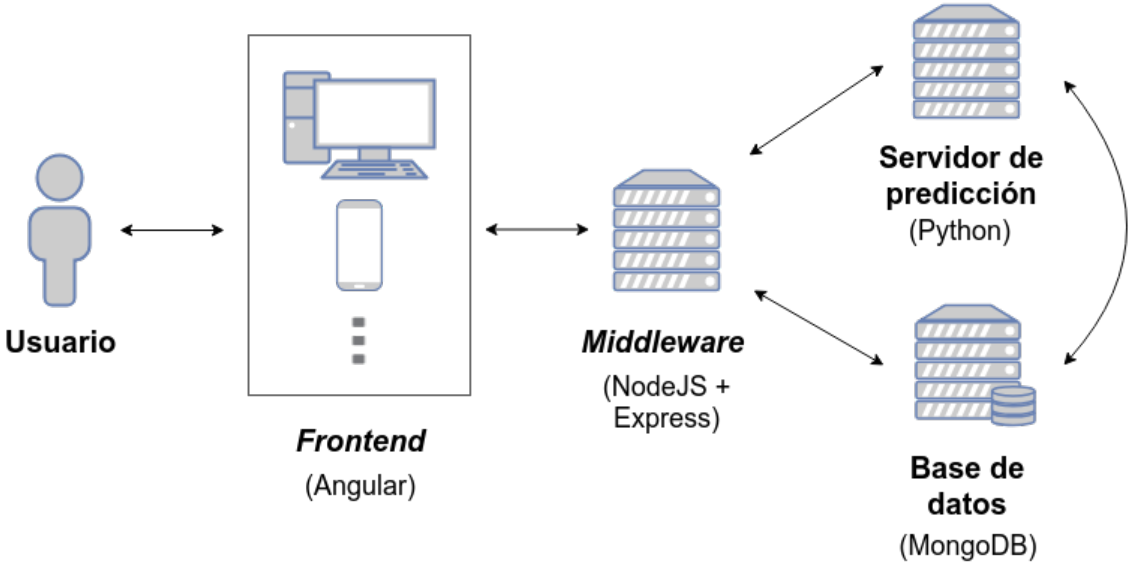

Figura 4.2: Arquitectura, comunicaciones y tecnologías del sistema

4.4 Componentes

4.4.1. *Frontend*

Función principal

El *frontend* es el punto de entrada del usuario en el sistema. Por tanto, es la parte en la que se produce toda la interacción usuario-sistema, lo cual implica que su función principal debe ser representar el sistema de una forma entendible para el usuario. Es decir, debe mostrar los datos del sistema de forma gráfica para que dicho usuario sea capaz de entender lo que está pasando y pueda realizar una correcta interacción con el sistema[4].

Tecnología

Existen varios *frameworks* actualmente para llevar a cabo esta tarea de forma más fácil y mantenible. En este caso, se ha usado una plantilla proporcionada por el ITI, la cual usa las siguientes librerías:

- Bootstrap[6], un *framework* que incluye modelos de componentes *web* así como extensiones del lenguaje JavaScript. También permite crear fácilmente páginas *web* adaptativas. Esto es, páginas web que se adaptan al dispositivo desde el cual se está accediendo
- JQuery[7], una librería para interactuar con los elementos del documento HTML de forma más sencilla
- Font Awesome[8], una librería de iconos e imágenes

Esta plantilla está basada en Angular 4, aunque existen otros *frameworks* similares como son VueJS[9] o React[10] que sirven para el mismo propósito.

Funcionamiento

El *frontend* se compone de un conjunto de pantallas por las que podrá navegar el usuario:

- La **pantalla de inicio de sesión**, la cual funcionará como punto de acceso a las demás, de forma que si el usuario no se identifica no puede acceder a las pantallas siguientes
- La **pantalla principal**, que consistirá en una página en la que se dispondrá de dos apartados:
  - El **apartado de gestión**, en el que por el momento solo se podrá dar de alta un nuevo paciente
  - El **apartado de predicción**, en el que se mostrarán disponibles las tareas de predicción

[6] https://getbootstrap.com/
[7] https://jquery.com/
[8] https://fontawesome.com/
[9] https://vuejs.org/
[10] https://reactjs.org/

- La **pantalla de tarea de predicción**, en la que se rellenarán los datos pertinentes y se obtendrá la predicción
- La **pantalla de nuevo paciente**, en la que se rellena un único campo que es el código identificador del paciente

A continuación, se describe el funcionamiento detallado de cada una de ellas, y cómo el usuario interactúa con ellas:

- La primera pantalla a la que se accede se trata de la pantalla de inicio de sesión. En ella se introduce el nombre de usuario y contraseña. Dependiendo de si estos son correctos se accede a la siguiente página o se muestra un mensaje de error indicando que los datos no son correctos
- La pantalla principal consiste en una página en la que se muestran todas las tareas de predicción disponibles. Para cada tarea se debe mostrar sobre qué enfermedad trata, el título de la tarea, y las posibles respuestas que puede dar. Cuando el usuario pulsa sobre una tarea se abre la pantalla de tarea. También hay un apartado de gestión en el que se puede dar de alta a nuevos pacientes
- En la pantalla de tarea de predicción se muestra la información sobre la tarea y el formulario asociado a la misma, junto con un selector de paciente al cual se le va a asignar la predicción una vez efectuada. En el formulario, cada entrada se trata de una variable relevante para la predicción. Además, existe un recuadro en el que se muestra la predicción una vez obtenida.
- En la pantalla de introducir nuevo paciente se muestra un formulario en el que se introduce el identificador del nuevo paciente. Al pulsar enviar se da de alta el mismo, asignándose al usuario que le ha dado de alta y apareciendo posteriormente disponible para seleccionar en la pantalla de tarea de predicción

Tanto la pantalla principal como la pantalla de tarea se desarrollarán de forma que sean reactivas. Esto es, las tareas de predicción que aparezcan en la pantalla principal se obtendrán directamente de la base de datos, de manera que cada vez que se acceda a esta pantalla se realizará una petición al *middleware* para obtener todas las tareas disponibles y así mostrarlas al usuario. En cuanto a la pantalla de tarea, se mostrarán los pacientes disponibles para el usuario actual y las variables de esa tarea de la misma forma, se realiza una petición al *middleware* de los datos tanto de los pacientes asignados al usuario como de las variables de la tarea y se muestran al propio usuario. La ventaja que tiene esto es que si se quiere añadir una nueva tarea de predicción, simplemente lo que se debe hacer es actualizar la base de datos con los nuevos datos de la tarea, sin tener que pasar por modificar el *frontend* para que esa nueva tarea esté disponible.

4.4.2. *Middleware*

Función principal

El *middleware* tiene como función principal responder con los datos solicitados a la petición recibida de un cliente. Para ello, se comunica con dos servidores: el servidor de predicción y el servidor de la base de datos.

El *middleware* es un servicio RESTful, lo que significa que tiene definida una interfaz, lo que se conoce como API REST, y cualquier cliente que conozca esa interfaz puede interactuar con él.

Tecnología

- Passport[11], que simplifica todo el proceso de autenticación de usuario y permite crear pasarelas de forma que si un usuario no está autenticado no puede acceder a ciertas rutas de la web
- JWT[12], que es un estándar[13] y se usará para generar y gestionar el token único que identifica a cada usuario
- Mongoose[14], para gestionar una base de datos MongoDB mediante modelos

Funcionamiento

- Desde el cliente llega una petición HTTP, según la cual y dependiendo de los datos que requiera, se conecta con la base de datos o el servidor de predicción
- El *middleware* conecta con el servidor de predicción, en caso de que la petición sea de predicción, enviándole los datos del formulario para que los procese. En caso de que no haya ningún error, el servidor de predicción responde con la predicción junto con un indicador de fiabilidad de la misma, que se reenvía al cliente para que lo muestre al usuario
- En caso de que la petición sea de datos, el *middleware* conecta con la base de datos, la cual responde con los datos pertinentes que se reenvían al cliente
- Se ha de controlar también el acceso a datos y a las predicciones comprobando si el cliente que envía las peticiones ha iniciado sesión y tiene permiso para disponer de estos datos. Esto se realiza haciendo uso de los mecanismos que proporciona Passport junto con la tecnología JWT

[11] http://www.passportjs.org/
[12] https://jwt.io/
[13] RFC 7519
[14] http://mongoosejs.com/

4.4.3. Servidor de predicción

Función principal

El servidor de predicción tiene como función proporcionar un servicio el cual responda con una predicción generada a partir de técnicas de ML en función de los datos que recibe.

Tecnología

En este caso, dadas las características del sistema y las muchas librerías existentes para este lenguaje de programación para el uso de técnicas de ML así como de la versatilidad y facilidad de aprendizaje del mismo, se ha escogido Python para implementar esta parte. Se usa la librería scikit-learn[15] junto con Numpy[16], que se utilizan como base para desarrollar esta parte. Además, también se usan librerías desarrolladas en el ITI que extienden la funcionalidad de scikit-learn y que facilitan la tarea de aplicación de técnicas de ML como el procesamiento de datos previo, la imputación de datos faltantes, etc.

Funcionamiento

El servidor de predicción, al ser un servidor aparte, es independiente de los demás componentes a excepción de la base de datos. Por lo tanto, cualquier sistema que sea capaz de comunicarse con él puede obtener una predicción. Para funcionar correctamente, el servidor necesita acceder a la base de datos, donde se guardan los modelos preentrenados mediante los cuales se realizarán las predicciones.

El esquema del flujo de trabajo que realizará el servidor es el siguiente:

- Al iniciarse, el servidor realiza una configuración inicial
- Después de esta configuración inicial, el servidor comienza un bucle infinito en el que espera a recibir una petición. Cuando recibe una, lo primero es comprobar si la sintaxis de la información recibida es correcta. En caso negativo, se descarta y se vuelve a esperar a recibir una nueva petición. Si es correcta, se realiza la predicción en base a los datos recibidos.
- Una vez realizada la predicción, se calcula un indicador de confianza para la misma y se responde con ambos elementos

[15] Librería que proporciona mecanismos para utilizar técnicas de ML, procesamiento de datos, imputación de datos faltantes, etc. (http://scikit-learn.org/stable/index.html)

[16] Librería fundamental para la computación científica (http://www.numpy.org/)

4.4.4. Base de datos

Función principal

La función principal de la base de datos es la persistencia. Es decir, debe mantener los datos guardados en alguna parte con el fin de que estén disponibles en cualquier momento que se requieran. Debe almacenar tanto las entidades del sistema (tareas, usuarios, etc.) como los modelos preentrenados que usará el servidor de predicción para realizar su función.

El esquema de la base de datos se corresponde con el diagrama de clases del sistema (figura 4.1).

Tecnología

La base de datos está construida con MongoDB, el cual usa un formato similar al que se usa para el intercambio de datos en el resto del sistema (JSON), lo que simplifica la interacción y gestión de los datos ya que, de esta manera, se guardan de forma muy similar a cómo se interpretan.

Funcionamiento

MongoDB dispone de un servicio el cual se ejecuta como un servidor que acepta peticiones y responde a las mismas con los datos requeridos, por lo que puede ubicarse en una máquina aparte que se dedique únicamente a persistir los datos y ser accesible desde el exterior.

CAPÍTULO 5
Implementación

En este capítulo se detalla la implementación realizada en base al diseño especificado en el capítulo anterior.

5.1 *Frontend*

5.1.1. Pantalla de inicio de sesión

Lo primero que se va a encontrar el usuario al acceder al sitio web es la pantalla de inicio de sesión. En esta pantalla, deberá introducir su nombre de usuario y su contraseña para acceder al sistema (figura 5.1).

Figura 5.1: Pantalla de inicio de sesión

Si el usuario introduce datos incorrectos o si no es posible conectar con el servidor se muestra un mensaje indicando de qué error se trata. En la figura 5.2 se ha introducido una contraseña incorrecta y se muestra un mensaje de error indicando que los datos no son correctos. En caso de que el servidor no estuviera disponible por cualquier motivo, en el mensaje de error se indicaría que no se puede establecer conexión con el servidor.

Figura 5.2: Error al iniciar sesión

5.1.2. Pantalla principal

Una vez que se ha iniciado la sesión satisfactoriamente, se muestra la pantalla principal (Figura 5.3), en la que se encuentran todas las tareas de predicción disponibles, además de un apartado de gestión en el que se puede dar de alta a un nuevo paciente.

Es importante destacar que el número de tareas de predicción disponibles es dinámico. Es decir, desde la base de datos se recuperan todas las tareas de predicción y se muestran, adaptándose al tamaño de la pantalla. Esto facilita más adelante el añadido de nuevas tareas, puesto que únicamente se han de añadir a la base de datos y no se requiere modificar la parte *frontend*. Por otro lado, como no es necesario obtener las variables de cada tarea en este momento, desde el servidor se reciben los datos mínimos de dichas tareas. Esto es, el título, la enfermedad asociada y las posibles respuestas.

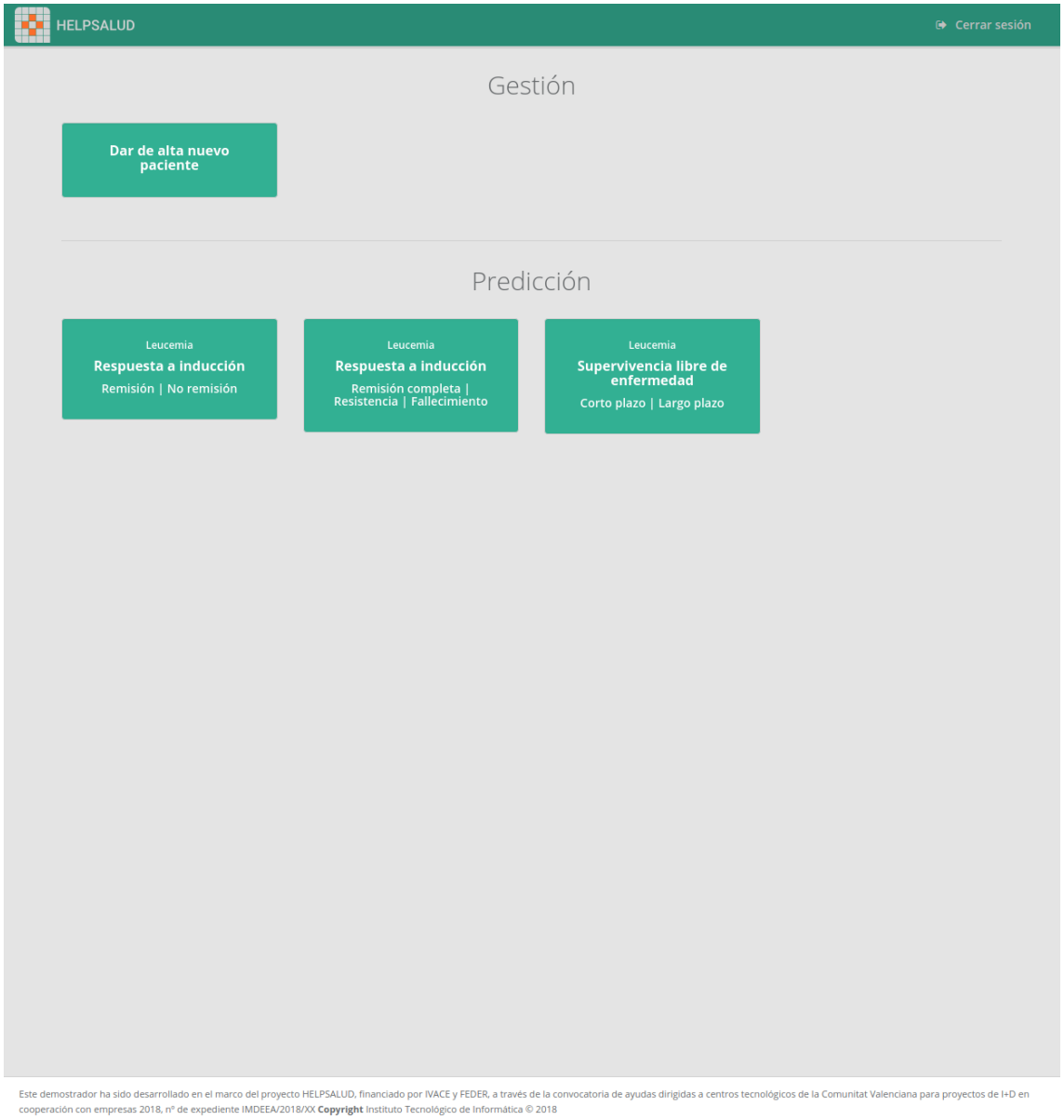

Figura 5.3: Pantalla principal

5.1.3. Pantalla de alta de nuevo paciente

Cuando el usuario elige dar de alta a un nuevo paciente, se muestra la pantalla de alta de nuevo paciente en la que aparece un formulario de un único campo que el usuario ha de rellenar con el código identificador del paciente. Cuando el usuario pulsa enviar, el paciente será añadido a la base de datos y se le asignará al usuario que le ha dado de alta.

Figura 5.4: Pantalla de dar de alta a un nuevo paciente

5.1.4. Pantalla de tarea

Cuando el usuario selecciona una tarea de predicción, se hace una petición al servidor para obtener toda la información relacionada con dicha tarea. una vez obtenida esta información, se muestra por pantalla, la cual está dividida en cuatro partes (ordenadas desde la parte superior de la pantalla hasta la parte inferior):

- La información relacionada con la tarea. En concreto, se muestra la enfermedad asociada, el título de la tarea y la descripción de la tarea
- La casilla de paciente, en la que se selecciona el paciente sobre el que se quiere hacer la predicción, el cual es uno de los pacientes asignados al usuario actual que se obtienen desde el servidor
- La casilla de datos, en la que aparecen todas las variables relacionadas con la tarea
- La casilla de predicción, en la que se muestra la predicción o un mensaje de error si no hay conexión con el servidor o si este devuelve un error
- El botón de enviar, que solo está activo si se ha seleccionado un paciente y, como mínimo, se ha rellenado un campo de datos

Dado que la predicción va asociada a un paciente, no es posible realizar una predicción sin seleccionar uno de ellos. Además, tampoco es posible obtener una predicción si no se ha rellenado ningún campo, ya que no es posible realizar una predicción sin datos. Sin embargo, sí es posible obtener la predicción si se ha escogido un paciente y al menos un campo se ha rellenado. De esta forma, los datos sin valor serán imputados y, por tanto, la predicción puede ser menos acertada.

Si el usuario pulsa enviar y no se han rellenado todos los campos, se muestra una alerta indicándoselo, permitiéndole cancelar el envío para así completar los campos que faltan o, en caso de que los desconozca, continuar y enviar los datos. Si escoge esta última opción los campos no rellenados se harán más visibles cambiando de color y apareciendo

un mensaje en cada uno de ellos indicando que no se ha rellenado, como se puede observar en la figura 5.6. Esto se hace porque, si el formulario es muy extenso, puede que el usuario no se dé cuenta de que se ha dejado algunos campos sin rellenar.

Después de enviar los datos, desde el servidor llega una predicción que contiene la etiqueta que representa la predicción y el indicador de confianza de la misma. Una tarea, como se ha descrito en el diagrama de clases del sistema (figura 4.1), tiene un atributo *prediction*, que consiste en un mapa en el cual las claves son las etiquetas de las posibles predicciones y cuyos valores son estas predicciones humanamente legibles. Con la ayuda de este mapa se obtiene el texto que representa la predicción de forma que el usuario lo entienda y dicho texto es el que se muestra en la casilla de predicción.

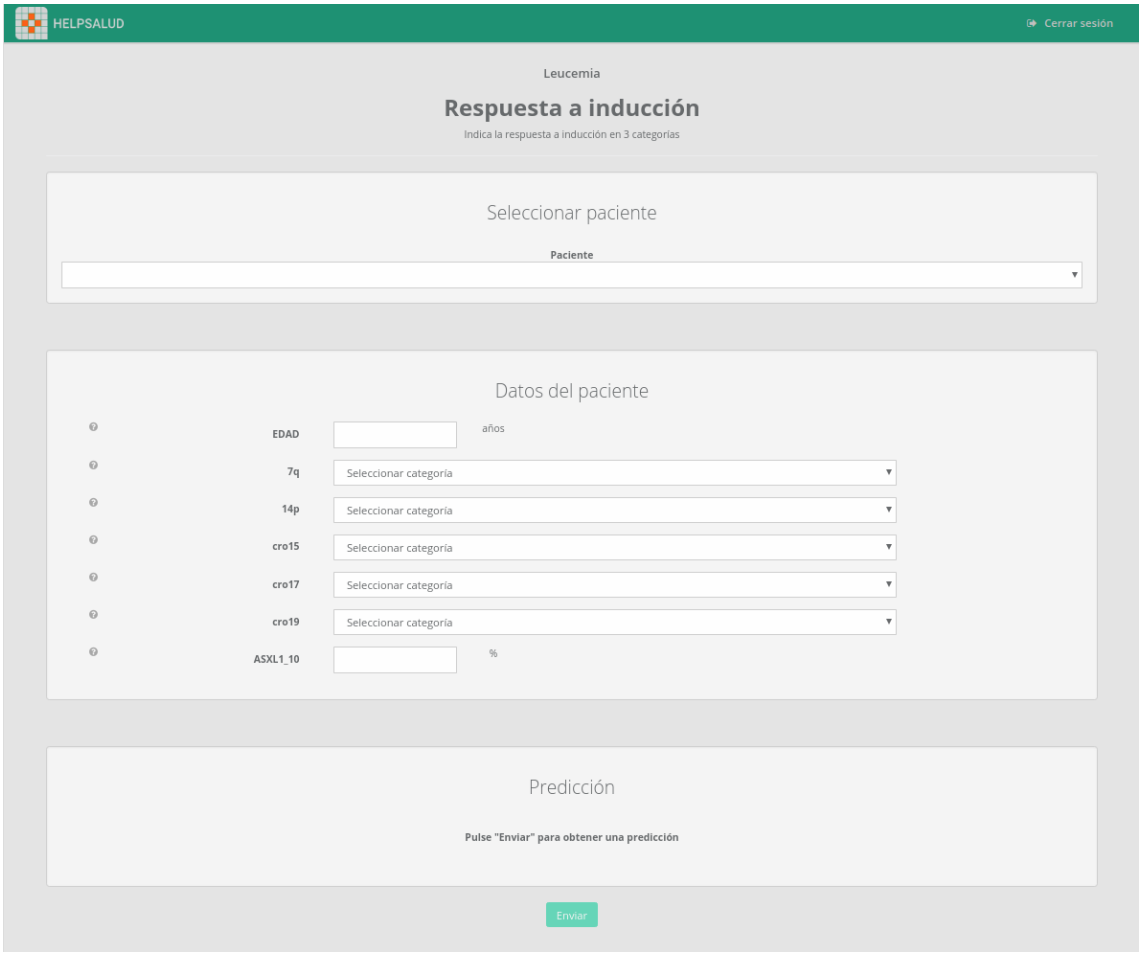

Figura 5.5: Pantalla de tarea antes de rellenar los datos

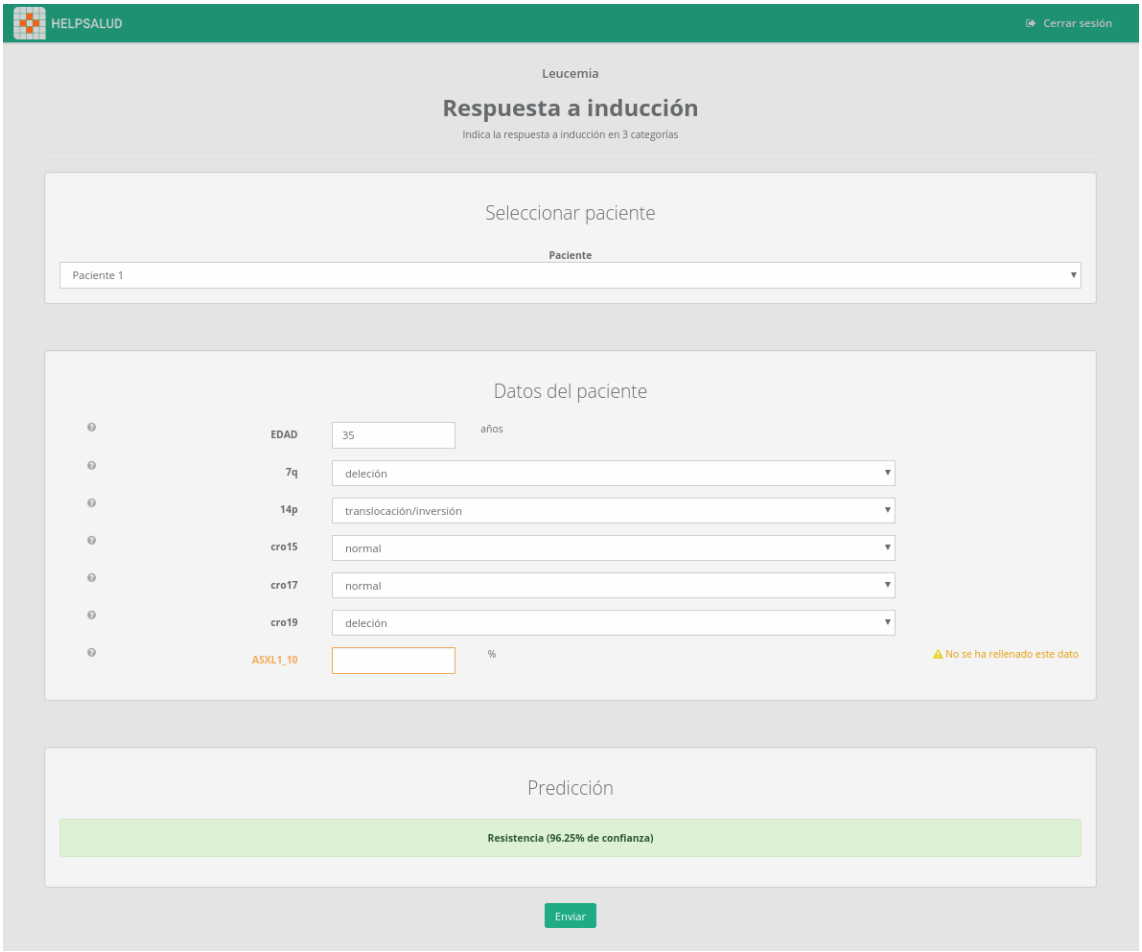

Figura 5.6: Pantalla de tarea después de enviar los datos, dejando uno sin valor

5.2 *Middleware*

Como se ha comentado en el capítulo anterior, el *middleware* debe definir una API REST que permita a los clientes interactuar con él. Esto es posible hacerlo de forma sencilla e intuitiva con Express.

Express se encarga de simplificar la creación de *endpoints* (puntos de entrada a una ruta definida de la API), proporcionando una interfaz simple para definirlos. A estos *endpoint* se les puede asignar una o más funciones de *middleware*, que se encargan de manejar la petición que ha llegado, de hacer algo con ella y de responder, o pueden delegar estas tareas en otras funciones de *middleware*. Cada función de *middleware* recibe tres parámetros, siendo el tercero opcional:

- *req*, que contiene información sobre la petición recibida
- *res*, que permite formular una respuesta a la petición recibida
- *next*, que permite llamar a la siguiente función de *middleware*. Se suele utilizar cuando la función actual ya ha terminado su tarea y deja paso a la siguiente

También se pueden definir *routers*, que funcionan como *middlewares* que permiten crear distintos *endpoint*. De esta forma, se pueden definir *routers* que definan nuevos *endpoint* y en los que se delegue las peticiones recibidas a un determinado *endpoint*.

Para definir la API, se definen cuatro rutas principales, para la que cada una de ellas se define un *router*:

- Auth, que se encarga del manejo del inicio de sesión
- Tasks, que se encarga de la parte de la interfaz que se ocupa de las tareas de predicción
- Patients, que se encarga de la parte de la interfaz que maneja los pacientes
- Features, que se encarga de la parte de la interfaz que maneja las características

Estas rutas, como se puede observar en el fragmento 5.1, a excepción de Auth, pasan a través de una función de *middleware*, proporcionada por passport, la cual comprueba si el que ha mandado la petición a ese *endpoint* es un usuario identificado. Si es así, se permite el paso y se deja al *router* que maneje la petición. En caso contrario, se deniega el acceso. Para más información en cuanto al funcionamiento de Passport, véase el anexo B

```
1  // Se importan las rutas
2  const auth = require('./routes/auth');
3  const tasks = require('./routes/tasks');
4  const patients = require('./routes/patients');
5  const features = require('./routes/features');
6
7  const express = require('express');
8  const app = express()
9
10 app.use('/helpsalud/api/login', auth);
11
12 app.use('/helpsalud/api/tasks',
13   passport.authenticate('jwt', {session: false}),
14   tasks);
15
16 app.use('/helpsalud/api/patients',
```

```
17    passport.authenticate('jwt', {session: false}),
18    patients);
19
20  app.use('/helpsalud/api/features',
21    passport.authenticate('jwt', {session: false}),
22    features);
```

Fragmento 5.1: Rutas en Express

Para facilitar el mantenimiento y la legibilidad del código estas rutas se han definido en archivos aparte. Como se puede observar en el fragmento 5.1, se encuentran en la carpeta *routes*. Por ejemplo, en el archivo *routes/tasks.js* se define un *router* al que se delegan todas las operaciones que se hagan en la ruta */helpsalud/api/tasks*:

```
1   const router = require('express').Router();
2
3   router.get('/', function (req, res) {
4    ...
5   });
6
7   router.post('/', function (req, res) {
8    ...
9   });
10
11  router.get('/:id', function (req, res) {
12   ...
13  });
14
15  router.post('/:id', function (req, res) {
16   ...
17  });
18
19  module.exports = router;
```

Fragmento 5.2: Esqueleto del archivo *tasks.js*

Así, por ejemplo, toda petición GET y POST que se haga sobre */helpsalud/api/tasks/* ejecutará la función de la línea 3 en el caso de GET y 7 en el caso de POST y toda petición GET y POST que se haga sobre */helpsalud/api/tasks/<**id**>*, siendo **id** el identificador de una tarea, ejecutará la función de la líneas 11 y 15 respectivamente.

5.2.1. Comunicación con el servidor de predicción

Cuando desde el *frontend* se hace un POST a la dirección */helpsalud/api/tasks/**<id>***, lo que hace la función de *middleware* asociada es transmitir esa información por el *socket* ZMQ al servidor de predicción.

```
1  const pyAddr = "tcp://localhost:3100";
2  const zmq = require('zeromq');
3  ...
4  router.post('/:id', function (req, res) {
5      const serverSocket = zmq.socket('req');
6
7      const taskId = req.params['id'];
8      req.body['task'] = taskId;
9
10     serverSocket.on('message', function (serverRes, errZmq) {
11         if (res.headersSent) {
12             return;
13         }
14         serverSocket.close();
15
16         if (errZmq) {
17             return res.status(401).json({
18                 ok: false,
19                 error: errZmq
20             });
21         }
22
23         try {
24           const reply = JSON.parse(serverRes.toString());
25           return res.status(200).json(reply);
26         } catch (err) {
27           return res.status(500).json({
28             ok: false,
29             error: err
30           });
31         }
32     });
33
34     res.setTimeout(2500, function () {
35         return res.status(504).json({
36             ok: false
37         });
38     });
39
40     serverSocket.connect(pyAddr);
41     serverSocket.send(JSON.stringify(req.body));
42 });
```

Fragmento 5.3: Detalle de la comunicación con el servidor de predicción

Como se puede observar en el fragmento 5.3, cada vez que llega una petición POST a este *endpoint*, se añade al JSON recibido en la petición el id de la tarea a la que se le asocian esos datos. Este JSON que contiene toda la información se envía a través del *socket* ZMQ al servidor de predicción, después de establecer una cuenta atrás de 2,5 segundos en la que si en ese tiempo no se recibe una respuesta, se notifica al cliente de un error del servidor. Cuando se recibe la respuesta, esta se reenvía al cliente. En caso de que se reciba una respuesta del servidor de predicción con una sintaxis JSON incorrecta (lo cual no debería ocurrir), se responde al cliente indicándole un error interno del servidor.

5.2.2. Comunicación con la base de datos

La comunicación con la base de datos se realiza por medio de la librería Mongoose, la cual permite modelar una base de datos construida con MongoDB y utilizar estos modelos para interactuar con ella. Los modelos, en este caso, se pueden definir en base al diagrama de la base de datos expuesto anteriormente (figura 4.1).

La conexión con la base de datos se establece al inicio mediante el método *connect()* de Mongoose, al cual se le pasa como parámetro la dirección IP del servidor de la base de datos y que devuelve una promesa mediante la cual podemos saber si se ha establecido la conexión satisfactoriamente o no. En este caso se ha decidido que si se devuelve un error al intentar establecer la conexión, este se muestre por pantalla y se finalice el proceso del *middleware*, puesto que no se pueden realizar las operaciones de extraer información de la base de datos si no se puede conectar con la misma. Así, se puede detectar la causa del error y solucionarlo.

Para definir un modelo en Mongoose, primero necesitamos crear un esquema. Un esquema es un objeto en el que se enumeran los atributos de los que consta una entidad y se especifica su tipo. Los modelos, al igual que las rutas, están definidas en ficheros aparte dentro de la carpeta *models* del proyecto, de tal forma que cada fichero contiene un modelo que es exportado. En consecuencia, al importar el fichero desde cualquier otro sitio se obtiene el modelo correspondiente, con el que se puede trabajar.

Por ejemplo, para la entidad *User* se ha creado un fichero *user.js*, en el que se define su esquema y finalmente se exporta el modelo para que pueda ser usado externamente:

```
1  const mongoose = require('mongoose');
2
3  const userSchema = new mongoose.Schema({
4      username: String,
5      password: String,
6      admin: { type: Boolean, default: false }
7  });
8
9  const User = mongoose.model('User', userSchema);
10
11 module.exports = User;
```

Fragmento 5.4: Detalle de la definición del modelo de la entidad *User*

Como se puede observar, en el fragmento 5.4 se define el esquema de *User*, el cual tiene como atributos un nombre de usuario, una contraseña y un campo indicando si es administrador, lo que se corresponde exactamente con el diagrama de clases, como se ha comentado anteriormente.

Una vez definido el modelo se puede importar desde un fichero externo y hacer uso de él. Por ejemplo, para obtener un usuario de la base de datos según su nombre de usuario, podemos hacer lo siguiente:

```
1  const User = require('./models/user');
2
3  User.findOne({ username: 'doctor'}, (err, user) => {
4      if (!err) {
5          return user;
6      }
7      // handle error
8      ...
9  });
```

Fragmento 5.5: Ejemplo de obtención de un usuario según su nombre de usuario

De la misma forma que se ha definido el modelo para *User* se realiza para las demás entidades que componen el sistema. Así, se define un modelo para cada una de estas entidades en un archivo aparte (de esta forma se facilita la legibilidad y mantenimiento del código) y se importa este archivo para acceder al modelo, el cual dispone de los métodos necesarios para realizar un correcto intercambio de información con la base de datos.

5.3 Servidor de predicción

El servidor de predicción calcula una predicción en base a los datos que le llegan desde el *middleware*. Entre los datos recibidos se encuentra el identificador de la tarea, que se usa para obtener la tarea a partir de la cual se obtiene el modelo de la base de datos que se usa para realizar la predicción. Para más detalles en cuanto a los datos recibidos por el servidor de predicción, véase el anexo A.1.2.

Al iniciarse el servidor, se inicializan dos elementos que serán usados durante la ejecución del mismo:

- Un socket ZMQ que escucha en el puerto 3100, por donde llegarán los datos para realizar las predicciones. Este paso puede provocar una excepción si el puerto 3100 ya está en uso
- Una lista que contiene los nombres normalizados de todas las variables ordenados de una forma concreta, necesaria para saber cómo ordenar las variables que se reciben, ya que no sabemos si se reciben ordenadas de la misma forma con la cual se han entrenado los modelos o no (en el *frontend* desarrollado, por ejemplo, no lo están). Esto es obligatorio puesto que a la hora de hacer una predicción sobre un modelo que se ha entrenado con unas determinadas características en un orden concreto se interpretarán los datos de forma errónea si las características a usar para hacer la predicción tienen un orden distinto. Para solucionar esto, se establece un orden global que se seguirá tanto para entrenar los modelos como para realizar las predicciones

Una vez completada la inicialización sin errores se empieza un bucle infinito el cual sigue los siguientes pasos, representados en la figura 5.7:

1. Se espera hasta recibir datos por el socket
2. Se comprueba que los datos recibidos estén en formato JSON o la sintaxis de este no sea incorrecta. En caso contrario se descartarán esos datos, se enviará un mensaje de error a través del socket y se volverá a esperar hasta recibir nuevos datos, descartando los recibidos
3. Se recupera de la base de datos el modelo preentrenado correspondiente a la tarea
4. Se aplican las transformaciones necesarias a los datos y, en caso de que haya algún dato faltante, se imputa mediante técnicas de *missing data* según el tipo de dato
5. Se realiza la predicción sobre el modelo, y además se calcula un indicador de confianza en la clasificación realizada en forma de porcentaje
6. Se envía la predicción obtenida y el indicador por el socket

Para esperar a recibir datos se usa el método *recv_json()*, un método de la librería de ZMQ, que bloquea el hilo de ejecución hasta que se reciben datos y que los procesa para devolver un objeto Python. En caso de que los datos recibidos no estén en formato JSON saltará una excepción de tipo JSONDecodeError. Si salta esta excepción se envía un mensaje de error a través del socket y se vuelve a esperar a recibir nuevos datos.

Como se explica en el anexo A.1.2, en cada JSON recibido debe existir un campo con el nombre de *task*, cuyo valor es el identificador de una tarea a la cual corresponden las variables también recibidas. A partir de este identificador podemos recuperar la tarea de la base de datos y, con ella, el modelo asociado, tal y como se puede observar en el diagrama de clases del sistema (figura 4.1).

Para las transformaciones de los datos, debemos tener en cuenta los tipos de dato con los que tratamos:

- Tipo numérico. Un valor numérico como puede ser la edad de una persona
- Tipo categórico. Un tipo de dato que tiene unos valores determinados, y el valor de una variable solo puede ser uno de ellos

Cada transformación de dato se delega en el módulo *feature*, módulo que contiene una clase abstracta *Feature* con un método abstracto *transform()*, y que de ella heredan tantas clases como tipos de variable existen. Cada una implementa su método *transform()* de forma que siempre se devuelve el valor de la variable convertido a característica. El módulo proporciona mecanismos para leer a partir de un fichero automáticamente las variables, para asignarles un valor, etc. y es uno de los módulos que componen la librería *data_preparation* desarrollada en el ITI. De esta forma, a partir de los datos recibidos se obtienen de la base de datos toda la información de cada variable, a partir de la cual se construyen objetos *Feature*. Una vez se dispone de estos objetos, para aplicar las transformaciones simplemente se llama al método *transform()* antes mencionado.

La imputación de datos faltantes se realiza mediante la librería *missing_data*, desarrollada en el ITI, que contiene una serie de módulos que dan opción a la imputación del valor de una variable a partir de las demás muestras, y además hace posible realizar la imputación a partir de un regresor con una determinada configuración o usar uno predefinido para cada método. Por tanto, para imputar algún valor faltante, se llama al método *missing_value_impute()*, al que se le pasa como parámetros los valores de las variables recibidas y el conjunto de valores de entrenamiento, que se recupera junto con el modelo de la base de datos.

Para la predicción se usa el método *predict()* del clasificador, que realiza la predicción y devuelve el resultado. Después, se usa el método *predict_proba()* para calcular el indicador de confianza de la predicción. Acto seguido, se envían a través del socket y el bucle comienza de nuevo.

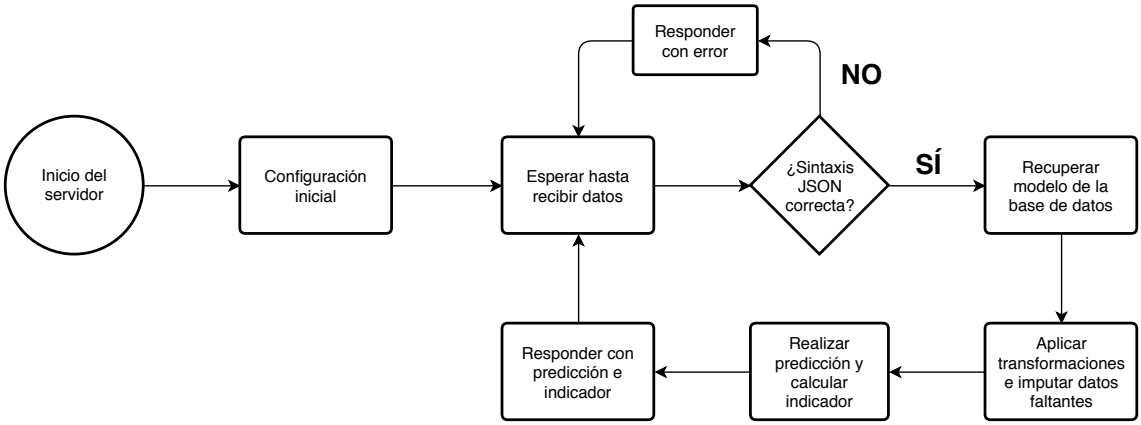

Figura 5.7: Flujo de trabajo del servidor de predicción

5.4 Base de datos

El tratamiento de la base de datos durante el desarrollo se ha realizado de dos formas:

- El uso de la versión gratuita de la herramienta NoSQLBooster[1], la cual proporciona una interfaz gráfica muy intuitiva para visualizar los datos guardados, añadir nuevos, eliminar, etc.
- El uso de un módulo Python que se ha desarrollado para acceder a la base de datos del sistema. Este módulo, llamado *database*, funciona como un servicio que se conecta a la base de datos y ofrece métodos CRUD específicos de cada entidad del sistema (*Task*, *Variable*, *Patient*...). Además, establece una interfaz para interactuar con la base de datos directamente mediante los métodos definidos en la librería PyMongo[2], la cual este módulo usa internamente para establecer la conexión con la base de datos

El módulo *database* puede ser usado a partir de ahora para añadir nuevas tareas de predicción, modificar la base de datos, etc. de forma sencilla.

[1] https://nosqlbooster.com
[2] librería oficial de MongoDB para Python (https://api.mongodb.com/python/current/)

CAPÍTULO 6
Implantación

Para la implantación del sistema es necesario una máquina accesible desde cualquier parte, es decir, que no esté en una red interna, en la que esté el *middleware* esté operativo, ya que será el que acepte las conexiones desde el exterior. También se requiere de otra máquina en la que implantar el servidor de predicción, además de la base de datos. Como se ha comentado ya, estos tres componentes son independientes entre sí, por lo que no necesitan estar en una misma máquina. Es decir, pueden distribuirse.

Para esto, el ITI dispone de un *cluster* de máquinas virtuales usadas para realizar cómputos, por lo que se ha tomado la decisión de implantar el sistema en estas máquinas virtuales, aunque para este caso y por limitación de recursos, se implantará todo el sistema en la misma máquina, dado que los componentes del mismo no consumen excesivos recursos de cómputo. Estas máquinas no son accesibles desde el exterior. Por eso, el Departamento de Sistemas de la empresa tuvo que hacerse cargo de redirigir el tráfico externo a esa máquina.

En cuanto a los aspectos técnicos, hay que tener en cuenta:

- Angular, mientras está en desarrollo, tiene su propio servidor interno para que se pueda visualizar lo que se va implementando. Cuando se termina el desarrollo, se hace un *build* en modo producción, de forma que genera una carpeta *dist* en la que están todos los *scripts* y archivos necesarios para que se muestre correctamente la página *web*

- Una vez tenemos el *build* de Angular hecho, el servidor de NodeJS (el *middleware*) debe apuntar a la carpeta *dist* antes mencionada, para que de esta forma, cuando se haga una solicitud, se responda a la misma enviando la página HTML asociada junto con los *scripts* y los estilos correspondientes, anteriormente generados por Angular

- Se ha de configurar el puerto en el que el *middleware* va a escuchar. Se ha escogido el puerto 3000, aunque desde el exterior es transparente, pues se ha implementado también un proxy inverso en la máquina de forma que la redirección que realiza el Departamento de Sistemas llegue siempre a un determinado puerto y, mediante este proxy inverso, se redirija al 3000 o al que se quiera en un futuro

- En cuanto al servidor de predicción, se debe configurar el puerto en el que se va a escuchar. Se ha escogido el puerto 3100, de forma que el *middleware* u otro solicitante, si lo hubiera, deberá conectarse mediante un *socket* ZMQ a la máquina en la que esté funcionando el proceso del servidor de predicción y a ese puerto, enviando los datos necesarios

- La base de datos también se implanta en la misma máquina, y no es necesario configurarla. Lo único que se ha de hacer es instalar el servicio de base de datos de MongoDB para poder acceder a ella. Una vez, instalada, se crea una nueva base de datos, la cual será la que se use tanto para los datos del sistema como para los modelos preentrenados

En definitiva, para la implantación se hace el *build* en producción de Angular y en el *middleware* se apunta la carpeta para cualquier petición que llegue, se configura la dirección a la que apuntará el socket ZMQ (en este caso en la misma máquina en el puerto 3100) y en el servidor de predicción se configura para que escuche en la misma máquina en el puerto 3100. Además, se crea una nueva base de datos que se usará para la persistencia del sistema.

CAPÍTULO 7
Conclusiones

Podemos concluir que se han cumplido en gran medida los resultados en cuanto a los objetivos establecidos del proyecto. En primera instancia, teníamos como caso de uso la enfermedad de la Leucemia Mieloide Aguda, para la que se esperaba, como mínimo, poder generar una predicción en cuanto a la respuesta al tratamiento de inducción en dos categorías: Remisión o no remisión de la enfermedad. Dado el rápido desarrollo del proyecto ha sido posible la creación de dos nuevas tareas de predicción, aparte de la ya planeada. Una de ellas realiza una predicción, también, en cuanto a la respuesta del paciente al tratamiento de inducción asignado pero en tres categorías: Remisión completa de la enfermedad, resistencia al tratamiento y fallecimiento del paciente. La otra realiza una predicción en cuanto a la supervivencia del paciente libre de enfermedad en un periodo de dos años desde la finalización del tratamiento en dos categorías: Corto y largo plazo.

En cuanto a los objetivos concretos del proyecto, estos han sido los resultados:

- Por un lado, se ha desarrollado satisfactoriamente un sistema *web* completo, esto es, se ha realizado un trabajo de *Full Stack Developer*, es decir, que se ha desarrollado la parte cliente y la parte servidor, además de todos los componentes del sistema y la comunicación entre ellos desde cero.
- Por otro lado, se ha desarrollado el componente de servidor de predicción, que a su vez forma parte del sistema desarrollado, el cual consigue realizar predicciones de forma satisfactoria, siendo la confianza de la predicción dependiente únicamente de la calidad de los modelos preentrenados de los que se disponga. En este caso, se disponía de modelos preentrenados del ITI según los criterios definidos en base a sus investigaciones previas sobre las variables relevantes de cada enfermedad, por lo que se asume que son modelos bien entrenados

CAPÍTULO 8
Trabajo futuro

Aunque se puede afirmar que el proyecto ha sido un éxito, dado que se han cumplido los objetivos y alcance establecidos, siempre hay mejoras que se pueden realizar. Por ejemplo, algunas de estas mejoras podrían ser las siguientes:

En primer lugar, sería muy adecuado separar el sistema existente ahora mismo en dos partes: Una parte consistiría en el apartado *web*, que consistiría en el *frontend*, el *middleware* (aunque en este punto sería más indicado hablar de servidor *web*) y la base de datos. La segunda parte consistiría en el servidor de predicción como un servicio independiente del sistema, de forma que esta aplicación *web* o cualquier otra aplicación puedan conectarse al mismo para realizar predicciones sin problemas. Así, conseguimos una separación del mundo *web* y el mundo de las técnicas de ML, de forma que pueda haber un equipo especialista *web* trabajando en la aplicación mientras haya un equipo experto en técnicas de ML que se encargue de mantener el servidor de predicción junto con los modelos.

Por otra parte, el sistema de almacenamiento de los modelos predictivos no es el más adecuado. En este momento, recordemos, los modelos se guardan en la base de datos de forma serializada. Esto no es un problema ahora mismo dado que dichos modelos no son demasiado grandes, pero según vayan aumentando su tamaño requerirán cada vez más capacidad de almacenaje y será más costoso recuperarlos de la base de datos y trabajar con ellos. Por tanto, como sistema alternativo y unido a la anterior mejora, dado que la base de datos ya no almacenaría los modelos, lo que mejoraría la separación entre las dos partes mencionadas en el punto anterior, se propone almacenar dichos modelos directamente en disco, serializados, ya que hay métodos existentes y eficientes para ejecutar esta acción en Python. Una forma de realizar esto pasaría por, por ejemplo, la creación de una carpeta llamada *models*, dentro de la cual exista un conjunto de subcarpetas donde el nombre de cada subcarpeta coincida con el identificador de la tarea asociada y que dentro de cada una de ellas se encuentre el modelo correspondiente. De esta forma, no se perdería la referencia tarea-modelo.

En cuanto a mejoras específicas del sistema *web*, por un lado, el apartado de gestión de la página *web* debería ser visible solo por los usuarios administradores. Además a este apartado de gestión se le deberían añadir más tareas de este tipo, como por ejemplo dar de alta a un nuevo usuario, pues ahora mismo solo existe la posibilidad de añadir un nuevo paciente a la base de datos. Por otro lado, como mejora de esta tarea, se propone cambiar el formulario actual de la misma por otro en el que se introduzca el código del paciente, como hasta ahora, y además se escoja el usuario al que se le quiere asignar el paciente de una lista de todos los usuarios que existen en el sistema. Las implementaciones descritas en este punto no se han realizado porque no entraban dentro de los objetivos del proyecto, y quedaban totalmente fuera del alcance del mismo, pero sí que pueden ser añadidas en versiones posteriores.

A nivel estético, la pantalla principal debería constar quizá de un menú de opciones, entre los que se encuentren la opción de gestión y de predicción, además de otras secciones que se puedan añadir. Al pulsar sobre cualquiera de estos nuevos apartados se pasaría a una siguiente pantalla en la que aparecerían todas las tareas disponibles de ese apartado. Por otra parte, antes de acceder a la pantalla de inicio de sesión sería indicado introducir una pantalla de bienvenida que explique un poco qué se va a encontrar el usuario, cómo funciona la aplicación, noticias, etc.

En cuanto a más funcionalidades, resultaría interesante añadir un apartado de historial, en el que el médico pueda revisar las predicciones realizadas a cada paciente y los datos de cada uno de ellos que se introdujeron, junto con la predicción que se obtuvo. También se podría añadir alguna opción para que, de alguna forma, el médico pueda introducir información de supervisión en el sistema, es decir, indicar su opinión acerca de una predicción realizada, como, por ejemplo, si le parece que estuvo acertada o no, y por qué. Este nuevo apartado tendría un efecto ventajoso tanto para los investigadores del ITI como para el médico, pues mediante esta nueva opción a los médicos les resultaría más sencillo realizar el seguimiento a sus pacientes, y a los investigadores del ITI les interesa saber si el modelo está bien entrenado, el porcentaje de acierto de las predicciones, etc. para poder seguir mejorando los modelos. Este apartado no se ha incluido en el proyecto, de nuevo, porque no estaba dentro del alcance del mismo, aunque la estructura actual del sistema da la posibilidad de la inclusión de esta mejora sin que suponga mucho esfuerzo.

Para finalizar, se propone crear una fase de diversos testeos de la aplicación, realizando especial hincapié en la usabilidad de la herramienta, ya que los médicos finalmente son los que dictarán su opinión sobre esta. Estos testeos ya se han realizado sobre distintas funcionalidades de la herramienta, pero de una manera informal y no exhaustiva, aunque está previsto hacerlo de manera formal en colaboración con el departamento de calidad del *software* del ITI.

Bibliografía

[1] Regalado, A. Eugenics 2.0: We're at the Dawn of Choosing Embryos by Health, Height, and More. *MIT Technology Review* [en línea]. 2017 [Consulta: abril de 2018] Disponible en https://www.technologyreview.com/s/609204/eugenics-20-were-at-the-dawn-of-choosing-embryos-by-health-height-and-more/

[2] Barry D. Representational State Transfer (REST) [en línea]. *Service Architecture*, 2000-2018 [Consulta: abril de 2018] Disponible en https://www.service-architecture.com/articles/web-services/representational_state_transfer_rest.html

[3] Colaboradores de Wikipedia. Representational State Transfer [en línea]. *Wikipedia, La enciclopedia libre*, 2005 [Consulta: abril de 2018]. Disponible en https://en.wikipedia.org/wiki/Representational_state_transfer

[4] Colaboradores de Wikipedia. Front-end web development [en línea]. *Wikipedia, La enciclopedia libre*, 2015 [Consulta: abril de 2018]. Disponible en https://en.wikipedia.org/wiki/Front-end_web_development

[5] Hasan N. MEAN App tutorial with Angular 4 [en línea]. *Medium*, 2017 [Consulta: abril de 2018]. Disponible en https://medium.com/netscape/mean-app-tutorial-with-angular-4-part-1-18691663ea96 y en https://medium.com/@nomanbinhussein/mean-app-tutorial-with-angular-4-part-2-4250522c845

[6] Richard O. Duda, Peter E. Hart, David G. Stork. *Pattern Classification*. USA: JOHN WILEY & SONS, 2001. ISBN 0-471-05669-3

[7] Maglogiannis I., Karpouzis K. *Emerging Artificial Intelligence Applications in Computer Engineering*. Ámsterdam: IOS Press, 2007. ISBN 978-1-58603-780-2

[8] Colaboradores de Wikipedia. Machine Learning [en línea]. *Wikipedia, La enciclopedia libre*, 2005 [Consulta: abril de 2018]. Disponible en https://en.wikipedia.org/wiki/Machine_learning

[9] Colaboradores de Wikipedia. Class diagram [en línea]. *Wikipedia, La enciclopedia libre*, 2006 [Consulta: abril de 2018]. Disponible en https://en.wikipedia.org/wiki/Class_diagram

[10] Colaboradores de Wikipedia. MEAN (software bundle) [en línea]. *Wikipedia, La enciclopedia libre*, 2015 [Consulta: mayo de 2018]. Disponible en https://en.wikipedia.org/wiki/MEAN_(software_bundle)

[11] Vanyan A. Learn using JWT with Passport authentication [en línea]. *Medium*, 2017 [Consulta: mayo de 2018]. Disponible en https://medium.com/front-end-hacking/learn-using-jwt-with-passport-authentication-9761539c4314

[12] Obermeyer Z., Emanuel E. Predicting the Future — Big Data, Machine Learning, and Clinical Medicine [en línea]. *NCBI*, 2016 [Consulta: junio de 2018] Disponible en https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5070532

[13] Humphries M. Missing Data & How to Deal: An overview of missing data [presentación de diapositivas]. *Texas Liberal Arts*, 2012 [Consulta: junio de 2018] Disponible en https://liberalarts.utexas.edu/prc/_files/cs/Missing-Data.pdf

APÉNDICE A
Contenido de los mensajes en las comunicaciones entre componentes del sistema

A.1 Peticiones

A.1.1. *Frontend* a *middleware*

Desde el *frontend*, cuando el usuario pulsa el botón Enviar, se crea un JSON en el que se introducen tanto los datos del formulario (nombre de la variable junto con su valor) como el identificador del paciente al que se le ha realizado la predicción. En el fragmento A.1 se puede observar un ejemplo de los datos comentados.

```
{
    "patient": "dnoom49jf2948f1613din2ug",
    "asxl1_1": 18.92497,
    "20p": 1,
    "monosomal": 2,
    ...
}
```

Fragmento A.1: Ejemplo de JSON enviado desde el *frontend*

A.1.2. *Middleware* a servidor de predicción

Desde el *middleware*, cuando se reciben los datos descritos en la sección A.1.1, se envía a través del *socket* ZMQ ese mismo JSON añadiendo el id de la tarea como nueva entrada y sustrayendo la entrada de la id del paciente, ya que el servidor de predicción no la necesita:

```
{
    "task": "5abcc9aa12771619a623acf4",
    "asxl1_1": 18.92497,
    "20p": 1,
    "monosomal": 2,
    ...
}
```

Fragmento A.2: Ejemplo de JSON enviado desde el *middleware*

A.2 Respuestas

A.2.1. *Middleware*

Cuando llega una petición se resuelve simplemente enviando los datos requeridos en formato JSON. Por ejemplo, en una petición de todas las tareas de predicción disponibles, se responderá con un JSON en el que hay un array en el que en cada posición hay una tarea. Si la petición es de una tarea concreta, se devolverán los datos de la tarea, etc. Se puede ver un ejemplo de respuesta a esta última petición en el fragmento A.3

```
{
    "variables": [(...)],
    "title":"Respuesta (...)",
    "description":"Indica (...)",
    "prediction":{(...)},
    "tag":"Leucemia",
    "name":"T1a_RI2",
    "model":"5ae2dbc7fa14e81f954c98ab",
    "id":"5abcc9aa12771619a623acf4"
}
```

Fragmento A.3: Respuesta a una petición GET de una tarea

Al ser un servicio RESTful, todas las peticiones que llegan al *middleware* lo hacen por medio del protocolo HTTP. Por tanto, no es necesario incluir dentro del JSON de respuesta si ha habido un error o no, ya que el propio protocolo proporciona los mecanismos para ello. No obstante, sí que puede estar incluido un campo de nombre *error* y de valor un mensaje explicativo del mismo en el JSON de respuesta, cuando el código de error HTTP usado no sea suficiente explicación.

En caso de que el usuario esté iniciando sesión, si los datos son correctos, se recibirá un JSON conteniendo el *token* JWT correspondiente además de un campo indicando si es administrador o no. Para más información, consúltese el anexo B

A.2.2. Servidor de predicción

Las respuestas siempre son en formato JSON. Este JSON siempre tendrá un campo de nombre *ok* con un valor booleano que indicará si ha habido un error. En caso de que haya habido un error, el valor del campo *ok* será *false* y también existirá un campo de nombre *error* que tendrá como valor un mensaje explicativo del error. En caso de que no haya ningún error, el valor del campo *ok* será *true* y existirán dos campos más: *res*, que contiene la predicción calculada, y *proba*, que contiene el indicador de fiabilidad de la predicción.

APÉNDICE B
Configuración de Passport

Passport proporciona diversos mecanismos para el control de la autenticación de los usuarios de forma muy sencilla y casi automática. Para ello, se ha de realizar una configuración inicial. Es decir, que método se va a seguir para identificar a los usuarios (lo que en Passport se llaman estrategias) e implementar el comportamiento que seguirán cada una de estas estrategias que se quieren implantar, que en este caso son dos.

En primer lugar, la ruta *Auth* se configura para que cuando le llegue un POST (el usuario envía sus datos) se utilice Passport mediante la estrategia llamada *local* para comprobar si estos datos recibidos pertenecen a un usuario existente. Si los datos son correctos, se genera un nuevo *token* JWT y se envía al usuario para que en las próximas peticiones se pueda identificar. Se puede observar esta configuración en el siguiente fragmento:

```
1  const router = express.Router();
2
3  const jwt = require('jsonwebtoken');
4  const passport = require('../passport/passport-config').passport;
5
6  const JWT_TOKEN = require('../passport/passport-config').JWT_TOKEN;
7
8  router.post('/', (req, res) => {
9
10     passport.authenticate('local', {session: false}, (err, user) => {
11         if (err || !user) {
12             return res.status(400).json({ok: false, error: err});
13         }
14
15         req.login(user, {session: false}, err => {
16             if (err) {
17                 res.json({ok: false, error: err});
18             }
19
20             const token = jwt.sign({id: user._id}, JWT_TOKEN);
21             return res.status(200).json({ok: true, token: token, admin:
                   user.isAdmin()})
22         })
23     })(req, res);
24 });
```

Para que esto funcione correctamente, en la configuración de Passport, se establece el comportamiento de la estrategia *local*, de forma que lo que llegue al *callback* de esta sea el nombre de usuario y la contraseña, recibidos en *Auth*. A partir de aquí se ha de definir, en función de cómo se quiera establecer el comportamiento de dicha estrategia, qué se quiere hacer para autenticar al usuario en función de los datos que se han recibido. En es-

te caso, se busca en la base de datos por el nombre de usuario, ya que es único y, si existe en la base de datos, se comprueba si la contraseña coincide. Se puede observar todo esto en el siguiente fragmento:

```
const User = require('../models/user');
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;

passport.use(new LocalStrategy(
    function(username, password, done) {
        User.findOne({ username: username }, (err, user) => {
            if (err) {
                return done(err);
            }

            if (!user) {
                return done(null, false);
            }

            if (!user.validPassword(password)) {
                return done(null, false);
            }

            return done(null, user);
        });
    })
);
```

La otra estrategia a usar consiste en la de JWT. JWT es un estándar RFC consistente en una cadena de caracteres codificada que dispone de 3 partes: una cabecera, una parte de datos y una firma de verificación. Dentro de la parte de datos se pueden introducir los datos que se quieran, de forma que quedan codificados con el algoritmo especificado en la cabecera, y se puede conocer si los datos están codificados correctamente mediante la firma de verificación.

En este caso, usaremos JWT para, en el apartado de datos, codificar el id del usuario que ha iniciado sesión. Después, se recibirá el *token* generado en la cabecera de cada una de las peticiones que se reciban. Por tanto, el proceso que sigue esta estrategia es el siguiente:

1. El usuario inicia sesión enviando sus datos al servidor
2. El servidor responde, en caso de que sean correctos, con un *token* JWT que contiene el identificador del usuario codificado
3. Cada vez que el usuario envíe una petición al servidor, en la cabecera de dicha petición se incluirá el *token* JWT que se ha recibido del servidor previamente. De esta forma, el servidor podrá reconocer de qué usuario se trata sin necesidad de que este introduzca sus datos de inicio de sesión de nuevo

Por tanto, lo único que se ha de hacer en cuanto a esta estrategia abarca es que cada vez que llegue una petición se compruebe si contiene el *token* en la cabecera. Si es así, se extrae el id del usuario mediante el cual se busca el usuario en la base de datos. En función de si se ha encontrado con éxito o no, se ejecuta un *callback* al que se le pasa el usuario si no ha habido ningún error, o el error, en caso de que haya habido alguno. El código asociado se puede observar en el siguiente fragmento:

```
const JWT_TOKEN = '06Y$U#&84gf72f34gqe#8nffBQ2W27m';

const User = require('../models/user');
const passport = require('passport');
const passportJWT = require('passport-jwt');
const ExtractJWT = passportJWT.ExtractJwt;
const JWTStrategy = passportJWT.Strategy;

passport.use(new JWTStrategy({
        jwtFromRequest: ExtractJWT.fromAuthHeaderAsBearerToken(),
        secretOrKey: JWT_TOKEN
    },
    (jwtPayload, cb) => {
        return User.findById(jwtPayload.id)
            .then(user => cb(null, user))
            .catch(err => cb(err));
    })
);
```

Mediante estas dos estrategias construimos el sistema de autenticación y posterior identificación de los usuarios que envían peticiones de una forma sencilla, rápida y eficaz.